

# 合肥工业大学

## 操作系统实验报告

实验题目	实验 4 线程的状态和转换
学生姓名	袁焕发
学 号	2019217769
专业班级	物联网工程 19-2
指导教师	田卫东
完成日期	2021 年 11 月 9 日

## 1. 实验目的和任务要求

调试线程在各种状态间的转换过程，熟悉线程的状态和转换。

通过为线程增加挂起状态，加深对线程状态的理解。

## 2. 实验原理

线程在其整个生命周期中（从创建到终止）会在多个不同的状态间进行转换。

当创建一个进程或线程时，新进程的主线程或者新线程都会被初始化为就绪状态，并被放入就绪队列中。当调度程序认为某个处于就绪状态的线程应当执行时，便使其成为当前运行的线程，该线程就会从就绪状态进入运行状态。当前运行线程因时间片用完或被更高优先级线程抢先时，当前运行线程就会由运行状态转入就绪状态。当前运行线程可能因调用 API 函数 `WaitForSingleObject` 等待事件或者执行 I/O 请求而被阻塞，从而由运行状态转入阻塞状态。处于阻塞状态的线程所等待的事件变为有效后，或等待超时时，该线程将被唤醒，从而由阻塞状态进入就绪状态。线程可以由任意一个状态转入结束状态。例如，线程执行完毕会由运行状态转入结束状态，就绪线程或者阻塞线程如果被强制结束，也会转入结束状态。

在引入挂起状态后，自然会增加挂起状态与其他各种非挂起状态之间的转换。一种最简单的转换情况是，从就绪状态（又称活动就绪）转入挂起状态（又称静止就绪），或者相反，当线程处于活动就绪状态时，使用挂起原语 `Suspend` 可以将该线程挂起，该线程便会转入静止就绪状态。处于静止就绪状态的线程，不会再被调度执行，直到使用原语 `Resume` 将该线程恢复为活动就绪状态。

## 3. 实验内容

### 3.1. 调试线程状态的转换过程

在本练习中，会在与线程状态转换相关的函数中添加若干个断点，并引导读者单步调试这些函数，使读者对 EOS 中的下列线程状态转换过程有一个全面的认识：

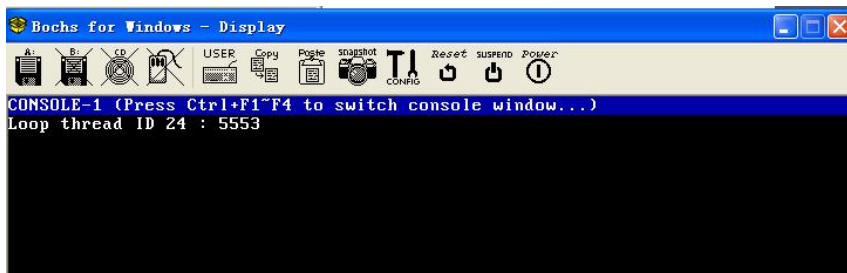
线程由阻塞状态进入就绪状态。

线程由运行状态进入就绪状态。

线程由就绪状态进入运行状态。

线程由运行状态进入阻塞状态。

按 F7 生成在本实验 3.1 中创建的 EOS Kernel 项目。按 F5 启动调试。待 EOS 启动完毕，在 EOS 控制台中输入命令“loop”后按回车，观察执行的效果，如下图所示。



接下来按照下面的步骤查看一下当 loop 线程正在运行时，系统中各个线程的状态，在 ke/sysproc.c 文件的 LoopThreadFunction 函数中，开始死循环的代码行（第 786 行）添加一个断点，启动调试，待 EOS 启动完毕，在 EOS 控制台中输入命令“loop”后按回车。打开“进程线程”窗口，可以查看当前系统中所有的线程信息，其中，系统空闲线程处于就绪状态，其优先级为 0；控制台派遣线程和所有控制台线程处于阻塞状态，其优先级为 24；只有优先级为 8 的 loop 线程处于运行状态，能够在处理器上执行。

数据源: POBJECT\_TYPE PspProcessType、POBJECT\_TYPE PspThreadType  
源文件: ps\psobject.c

进程列表

序号	进程 ID	系统进程 (System)	优先级 (Priority)	线程数量 (ThreadCount)	主线程ID (PrimaryThreadID)	镜像名称 (ImageName)
1	1	Y	24	7	2	'N/A'

线程列表

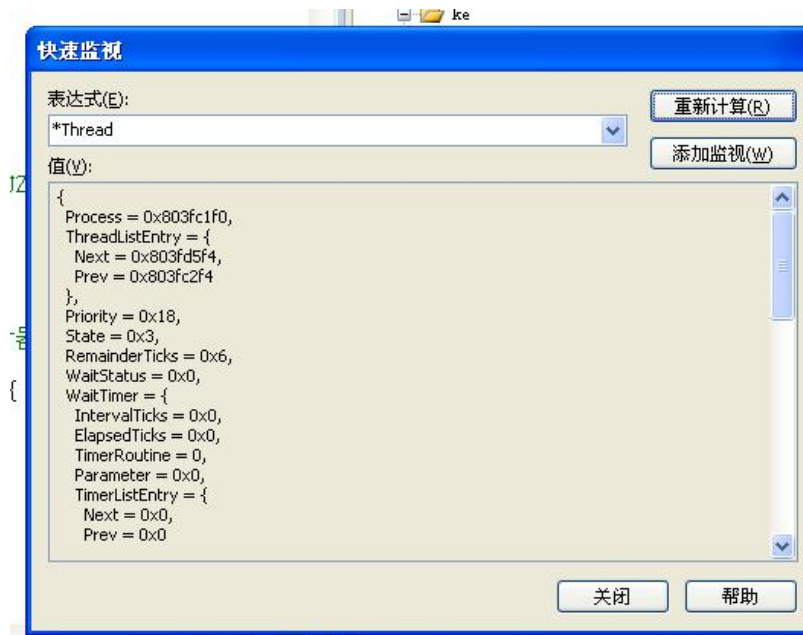
序号	线程 ID	系统线程 (System)	优先级 (Priority)	状态 (State)	父进程 ID (ParentProcessID)	起始地址与函数名 (StartAddress And FuncName)	
1	2	Y	0	Ready (1)	1	0x80017e40 KiSystemProcessRoutine	
2	17	Y	24	Waiting (3)	1	0x80015724 IopConsoleDispatchThread	
3	18	Y	24	Waiting (3)	1	0x80017f4b KiShellThread	
4	19	Y	24	Waiting (3)	1	0x80017f4b KiShellThread	
5	20	Y	24	Waiting (3)	1	0x80017f4b KiShellThread	
6	21	Y	24	Waiting (3)	1	0x80017f4b KiShellThread	
7	24	Y	8	Running (2)	1	0x80018a5c LoopThreadFunction	← PspCurrentThread

不要停止之前的调试，下面的步骤要在之前调试的基础上继续进行调试：删除所有断点，打开 ps/sched.c 文件，在与线程状态转换相关的函数中添加断点，在 PspReadyThread 函数体中添加一个断点（第 129 行），在 PspUnreadyThread 函数体中添加一个断点（第 161 行），在 PspWait 函数体中添加一个断点（第 226 行），在 PspUnwaitThread 函数体中添加一个断点（第 289 行），在 PspSelectNextThread 函数体中添加一个断点（第 402 行），按 F5 继续执行，然后激活虚拟机窗口。

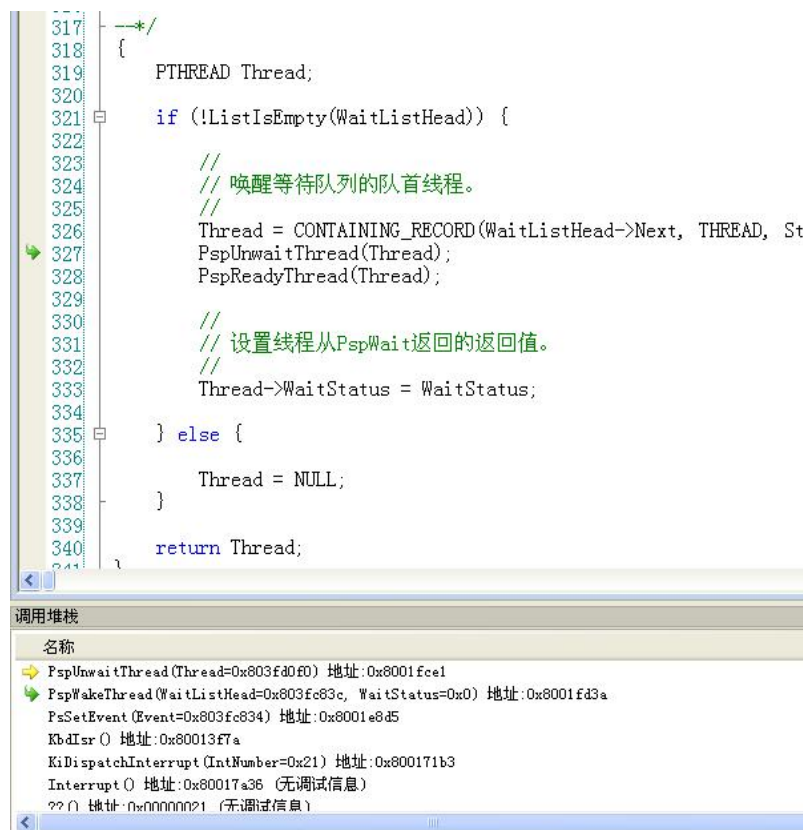
此时在虚拟机窗口中会看到 loop 线程在不停执行，而之前添加的断点都没有被命中，说明此时还没有任何线程的状态发生改变。

3.2. 控制台派遣线程由阻塞状态进入就绪状态

在虚拟机窗口中按下次空格键，此时 EOS 会在 PspUnwaitThread 函数中的断点处中断。在“调试”菜单中选择“快速监视”，在快速监视对话框的表达式编辑框中输入表达式“\*Thread”，然后点击“重新计算”按钮，即可查看线程控制块（TCB）中的信息。其中 State 域的值为 3（Waiting），双向链表项 StateListEntry 的 Next 和 Prev 指针的值都不为 0，说明这个线程还处于阻塞状态，并在某个同步对象的等待队列中；StartAddr 域的值为 IopConsoleDispatchThread，说明在 PspUnwaitThread 函数中要处理的线程就是控制台派遣线程。

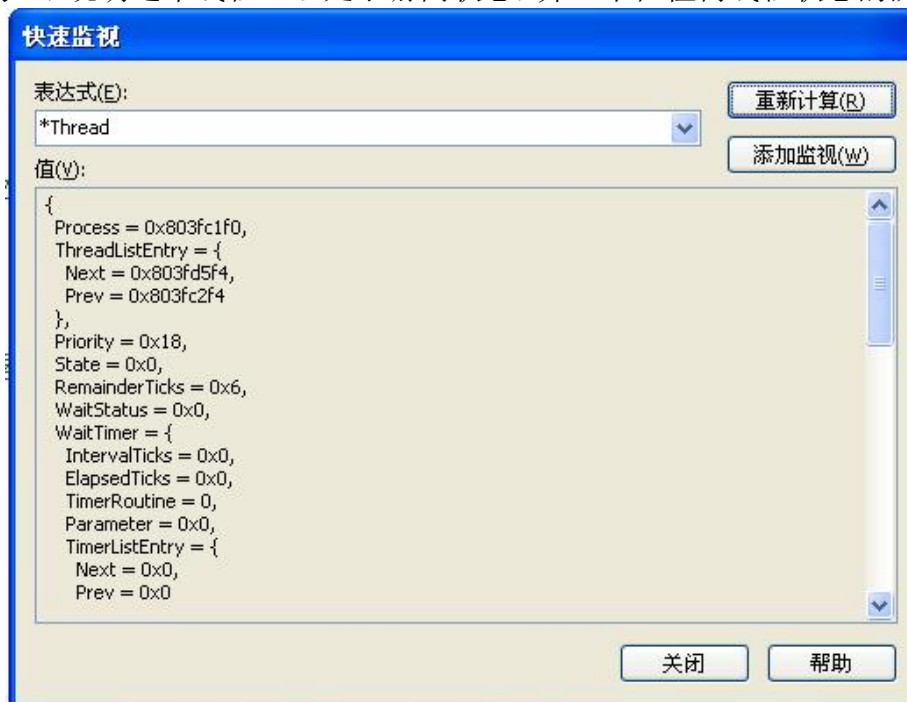


在“调用堆栈”窗口中双击 PspWakeThread 函数对应的堆栈项。可以看到在此函数中连续调用了 PspUnwaitThread 函数和 PspReadyThread 函数，从而使处于阻塞状态的控制台派遣线程先退出阻塞状态，然后再进入就绪状态。

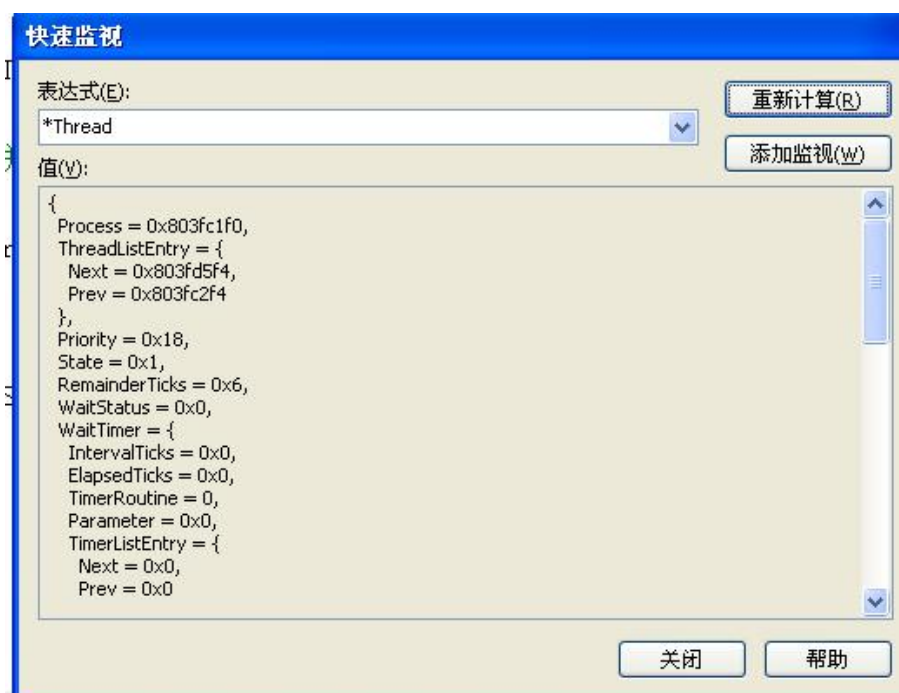


在“调用堆栈”窗口中双击 PspUnwaitThread 函数对应的堆栈项，再来看看此函数是如何改变线程状态的。按 F10 单步调试直到此函数的最后，然后再从快速监视对话框中观察“\*Thread”表达式的值。此时 State 域的值 0 (Zero)，双向链表项 StateListEntry 的 Next 和 Prev 指针的值

都为 0，说明这个线程已经处于游离状态，并已不在任何线程状态的队列中。



按 F5 继续执行，在 PspReadyThread 函数中的断点处中断。按 F10 单步调试直到此函数的最后，然后再从快速监视对话框中观察“\*Thread”表达式的值。此时 State 域的值为 1（Ready），双向链表项 StateListEntry 的 Next 和 Prev 指针的值都不为 0，说明这个线程已经处于就绪状态，并已经被放入优先级为 24 的就绪队列中。



选择“调试”菜单“窗口”菜单中的“就绪线程队列”菜单项，打开“就绪线程队列”窗口，在该窗口的工具栏上点击“刷新”按钮，可以看到在优先级为 24 的就绪队列中已插入线程 ID 为 17 的线程控制块，如图所示。

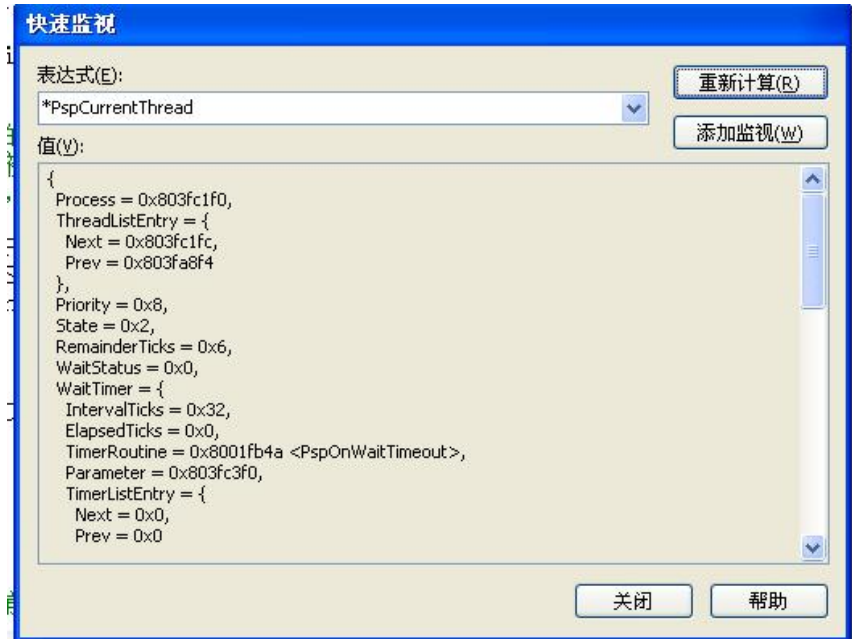
然后，在“线程运行轨迹”窗口，查看 ID 为 17 的线程的运行轨迹，可以看到该线程已由“阻塞”状态转化为“就绪”状态了。

Priority: 23	
Priority: 24	→
Priority: 25	
Priority: 26	
Priority: 27	
Priority: 28	
Priority: 29	
Priority: 30	
Priority: 31	

TCB	
Id	17
Priority	24
State	Ready
RemainderTicks	6
StartAddr	IopConsoleDispatchThread

3.3. loop 线程由运行状态进入就绪状态

按 F5 继续执行，在 PspSelectNextThread 函数中的断点处中断。在“快速监视”对话框中查看 “\*PspCurrentThread” 表达式的值，观察当前线程的信息。其中 State 域的值为 2 (Running)，双向链表项 StateListEntry 的 Next 和 Prev 指针的值都为 0，说明这个线程仍然处于运行状态。



按 F10 单步调试，直到对当前线程的操作完成（也就是花括号中的操作完成）。在“快速监视”对话框中查看 “\*PspCurrentThread” 表达式的值。其中 State 域的值为 1 (Ready)，双向链表项 StateListEntry 的 Next 和 Prev 指针的值都不为 0，说明 loop 线程已经进入了就绪状态，并已经被放入优先级为 8 的就绪队列中。刷新“就绪线程队列”窗口中，可以看到在优先级为 8 的就绪队列中已插入线程 ID 为 24 的线程控制块



象的引用。

快速监视

表达式(E):

\*PspCurrentThread

重新计算(R)

值(V):

```
{
  Process = 0x803fc1f0,
  ThreadListEntry = {
    Next = 0x803fc1fc,
    Prev = 0x803fa8f4
  },
  Priority = 0x8,
  State = 0x1,
  RemainderTicks = 0x6,
  WaitStatus = 0x0,
  WaitTimer = {
    IntervalTicks = 0x32,
    ElapsedTicks = 0x0,
    TimerRoutine = 0x8001fb4a <PspOnWaitTimeout>,
    Parameter = 0x803fc3f0,
    TimerListEntry = {
      Next = 0x0,
      Prev = 0x0
    }
  }
}
```

添加监视(W)

关闭

帮助

数据源: ULONG PspReadyBitmap, LIST\_ENTRY PspReadyListHeads[32]  
源文件: ps\sched.c



32个链表头组成的就绪队列(PspReadyListHeads[32])

Priority: 0

Priority: 1

Priority: 2

Priority: 3

Priority: 4

Priority: 5

Priority: 6

Priority: 7

Priority: 8

Priority: 9

Priority: 10

Priority: 11

Priority: 12

Priority: 13

Priority: 14

Priority: 15

Priority: 16

Priority: 17

Priority: 18

Priority: 19

Priority: 20

Priority: 21

Priority: 22

Priority: 23

Priority: 24

Priority: 25

Priority: 26

Priority: 27

Priority: 28

Priority: 29

Priority: 30

Priority: 31

TCB

Id

2

Priority

0

State

Ready

RemainderTicks

6

StartAddr

KiSystemProcessRoutine

TCB

Id

24

Priority

8

State

Ready

RemainderTicks

6

StartAddr

LoopThreadFunction

TCB

Id

17

Priority

24

State

Ready

RemainderTicks

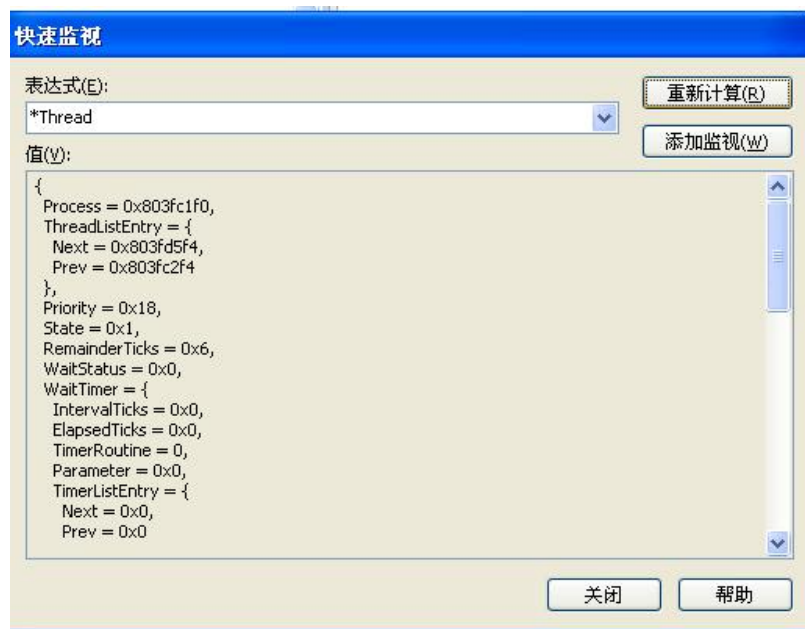
6

StartAddr

IopConsoleDispatchThread

### 3.4. 控制台派遣线程由就绪状态进入运行状态

按 F5 继续执行，在 PspUnreadyThread 函数中的断点处中断。在快速监视对话框中查看“\*Thread”表达式的值。其中 State 域的为 1（Ready），双向链表项 StateListEntry 的 Next 和 Prev 指针的值都不为 0，说明这个线程处于就绪状态，并在优先级为 24 的就绪队列中。

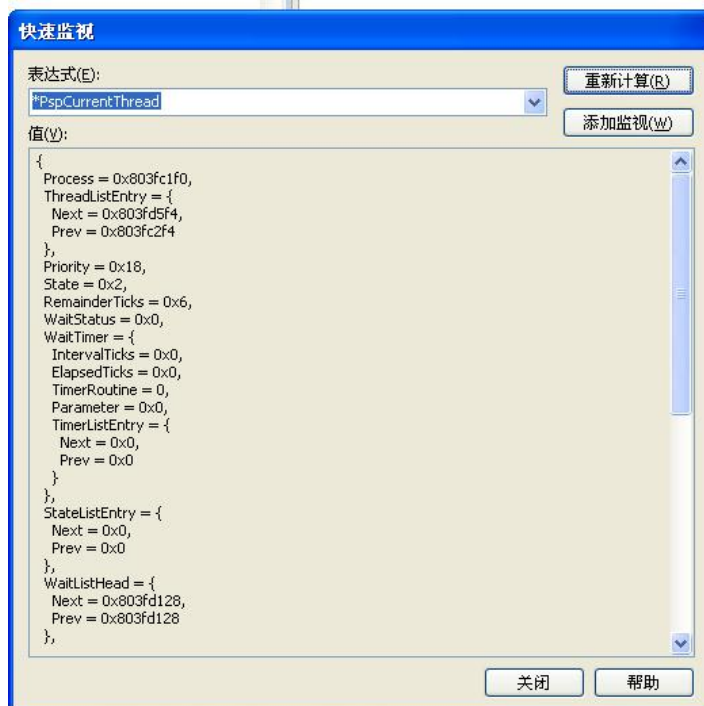


关闭快速监视对话框后，在“调用堆栈”窗口中激活 PspSelectNextThread 函数对应的堆栈项，可以看到在 PspSelectNextThread 函数中已经将 PspCurrentThread 全局指针指向了控制台派遣线程，并在调用 PspUnreadyThread 函数后，将当前线程的状态改成了 Running。



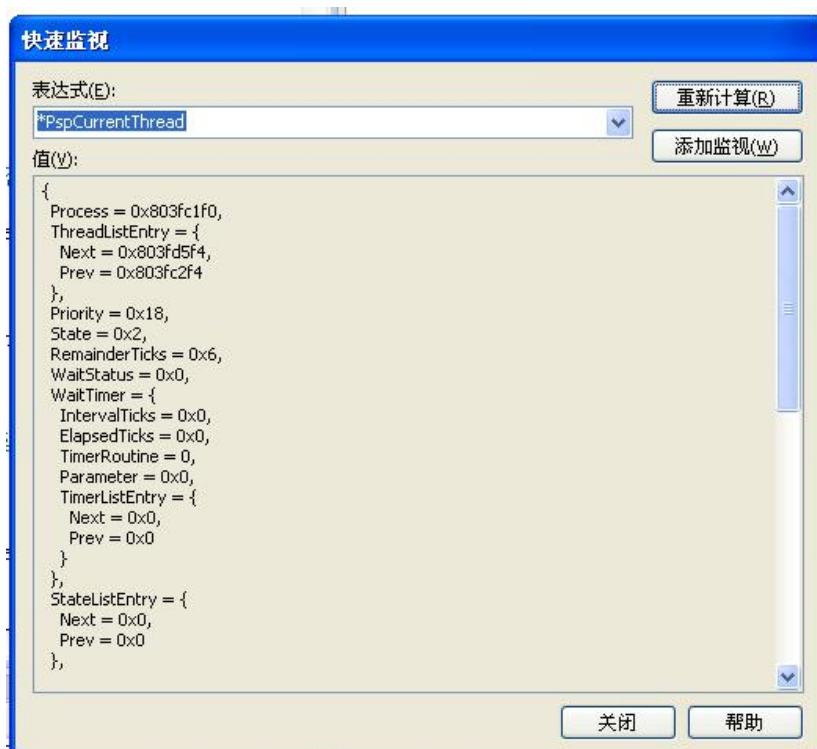
在“调用堆栈”窗口中激活 PspUnreadyThread 函数对应的堆栈项，然后按 F10 单步调试，直到返回 PspSelectNextThread 函数并将线程状态修改为 Running。再从快速监视对话框中查看“\*PspCurrentThread”表达式的值，观察当前占用处理器的线程的情况。其中 State 域的为 2（Running），双向链表项 StateListEntry 的 Next 和 Prev 指针的值都为 0，说明控制台派遣线程已经处于运行状态了。



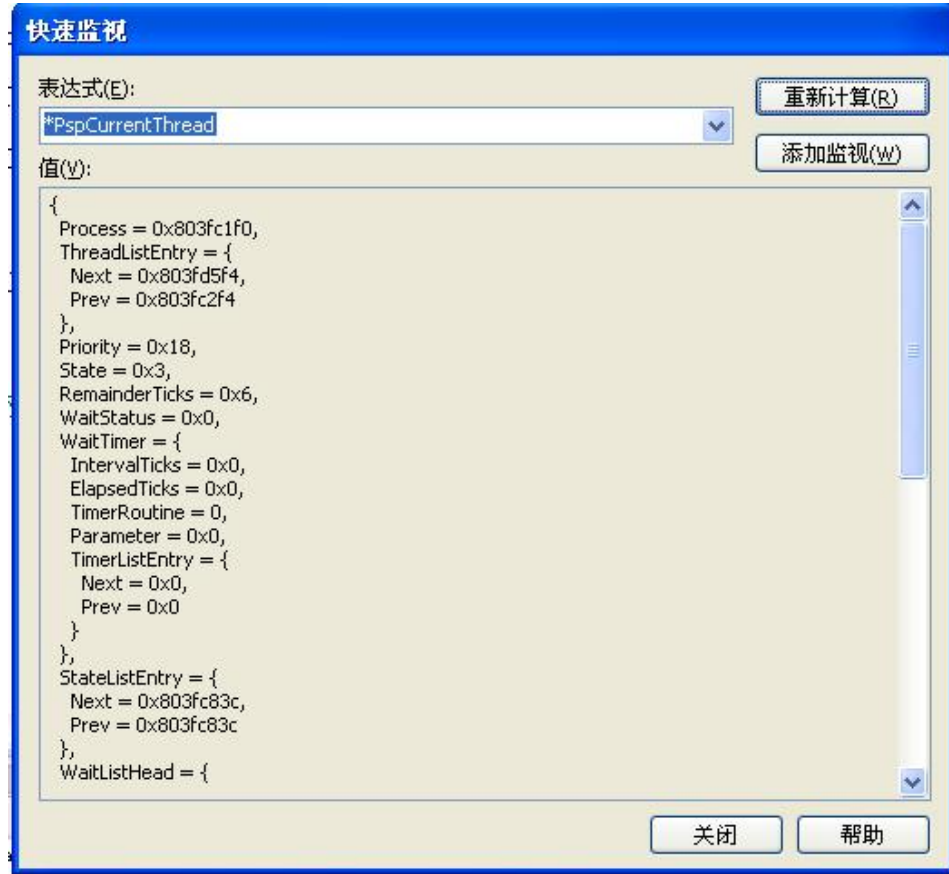


### 3.5. 控制台派遣线程由运行状态进入阻塞状态

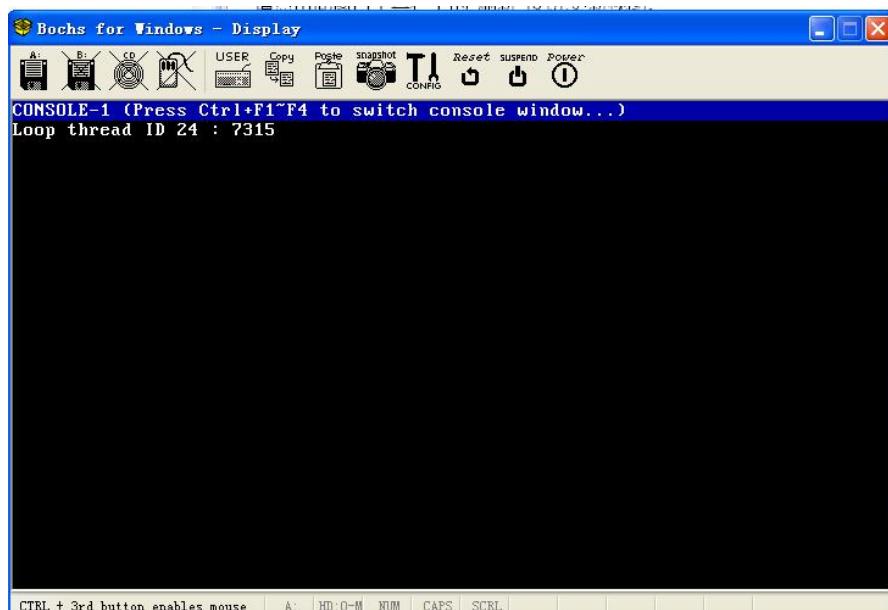
按 F5 继续执行，在 `PspWait` 函数中的断点处中断。在快速监视对话框中查看“`*PspCurrentThread`”表达式的值，观察当前占用处理器的线程的情况。其中 `State` 域的值为 2 (Running)，双向链表项 `StateListEntry` 的 `Next` 和 `Prev` 指针的值都为 0，说明这个线程仍然处于运行状态；`StartAddr` 域的值 `IopConsoleDispatchThread`，说明这个线程就是控制台派遣线程。



按 F10 单步调试，直到左侧的黄色箭头指向代码第 255 行。再从快速监视对话框中查看“\*PspCurrentThread”表达式的值。其中 State 域的值为 3 (Waiting)，双向链表项 StateListEntry 的 Next 和 Prev 指针的值都不为 0，说明控制台派遣线程已经处于阻塞状态了，并在某个同步对象的等待队列中。

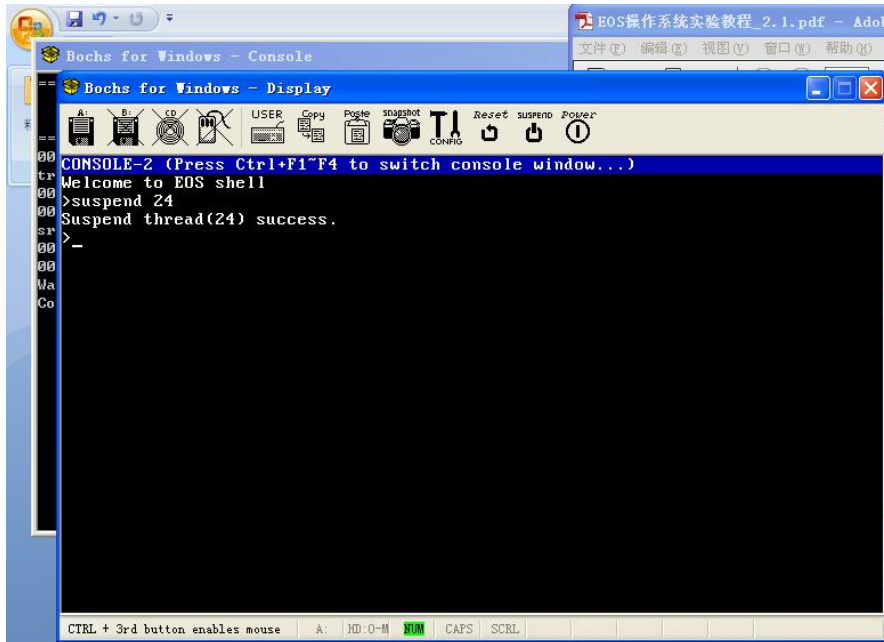


再按 F5 继续执行，EOS 不会再被断点中断。激活虚拟机窗口，可以看到 loop 线程又开始不停的执行死循环了。

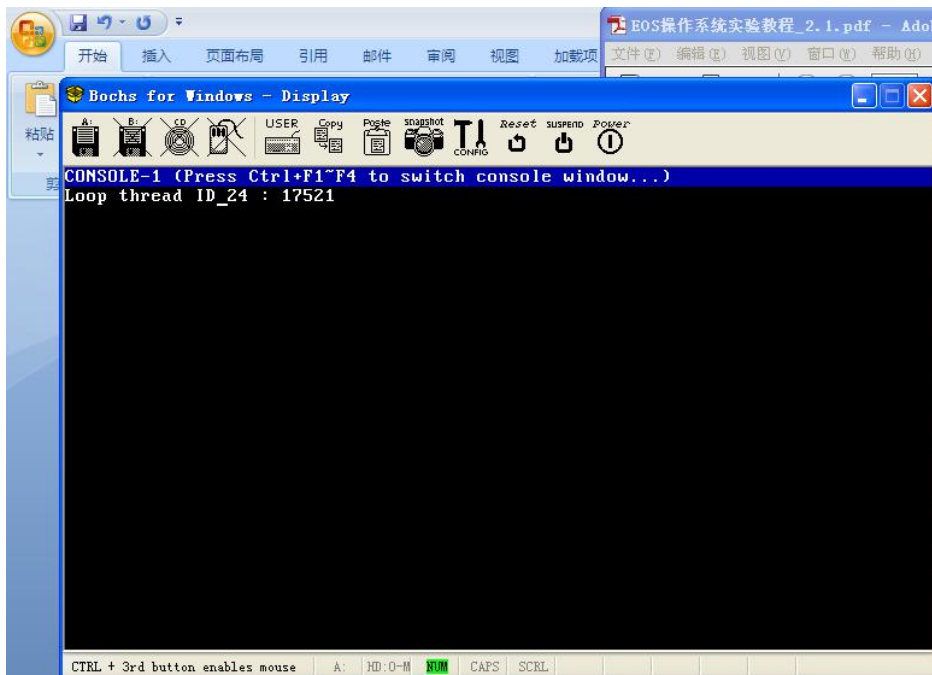


### 3.6. 为线程增加挂起状态

停止之前的调试过程，并删除之前添加的所有断点，按 F5 启动调试，待 EOS 启动完毕，在 EOS 控制台中输入命令“loop”后按回车。此时可以看到 loop 线程的执行计数在不停增长，说明 loop 线程正在执行。记录下 loop 线程的 ID。按 Ctrl+F2 切换到控制台 2，输入命令“suspend 24”（如果 loop 线程的 ID 是 24）后按回车。

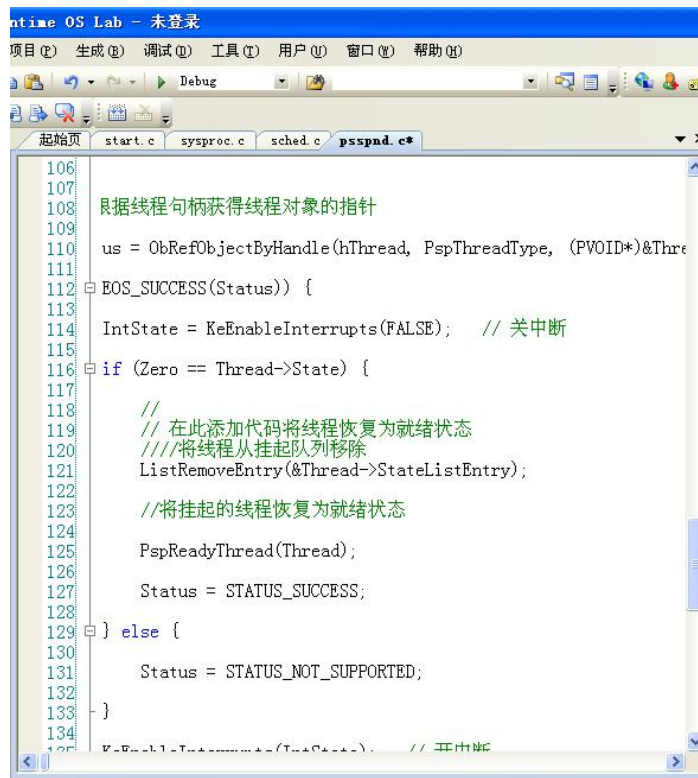


按 Ctrl+1 切换回控制台 1，可以看到由于 loop 线程已经成功被挂起，其执行计数已经停止增长了，此时占用处理器的是 EOS 中的空闲线程。



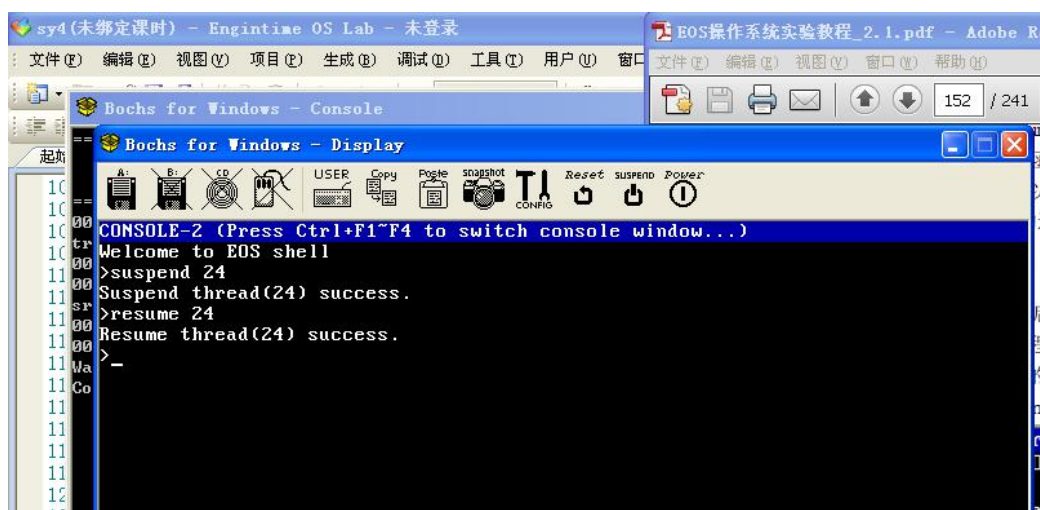
### 3.7. 要求及测试方法

Resume 原语可以将一个被 Suspend 原语挂起的线程恢复为就绪状态。但是 PsResumeThread 函数中的这部分代码（第 119 行）还没有实现，要求读者在这个练习中完成这部分代码。首先调用 ListRemoveEntry 函数将线程从挂起线程队列中移除，然后调用 PspReadyThread 函数将线程恢复为就绪状态，最后调用 PspThreadSchedule 宏函数执行线程调度，让刚刚恢复的线程有机会执行。



```
106
107
108 根据线程句柄获得线程对象的指针
109
110  us = ObRefObjectByHandle(hThread, PspThreadType, (PVOID*)&Thread);
111
112  EOS_SUCCESS(Status)) {
113
114      IntState = KeEnableInterrupts(FALSE); // 关中断
115
116      if (Zero == Thread->State) {
117
118          //
119          // 在此添加代码将线程恢复为就绪状态
120          ///将线程从挂起队列移除
121          ListRemoveEntry(&Thread->StateListEntry);
122
123          //将挂起的线程恢复为就绪状态
124
125          PspReadyThread(Thread);
126
127          Status = STATUS_SUCCESS;
128
129      } else {
130
131          Status = STATUS_NOT_SUPPORTED;
132
133      }
134
135      KeEnableInterrupts(TRUE); // 开中断
136
137  }
```

待读者完成 Resume 原语后，可以先使用 suspend 命令挂起 loop 线程，然后在控制台 2 中输入命令“Resume 24”（如果 loop 线程的 ID 是 24）后按回车。命令执行成功的结果如下图所示。如果切换回控制台 1 后，发现 loop 线程的执行计数恢复增长就说明 Resume 原语可以正常工作了。



## 4. 实验的思考与问题分析

- 4.1. 思考一下，在本实验中，当 loop 线程处于运行状态时，EOS 中还有哪些线程，它们分别处于什么状态。可以使用控制台命令 pt 查看线程的状态或者在“进程线程”窗口中查看线程的状态。

```
CONSOLE-2 (Press Ctrl+F1~F4 to switch console window...)
Welcome to EOS shell
>pt
***** Process List (1 Process) *****
ID : System? : Priority : ThreadCount : PrimaryThreadID : ImageName
1           Y          24           7           2           N/A

***** Thread List (7 Thread) *****
ID : System? : Priority : State : ParentProcessID : StartAddress
2           Y          0         Ready : 1              0x80017E40
17          Y          24         Waiting : 1              0x80015724
18          Y          24         Waiting : 1              0x80017F4B
19          Y          24         Running : 1              0x80017F4B
20          Y          24         Waiting : 1              0x80017F4B
21          Y          24         Waiting : 1              0x80017F4B
24          Y          8         Ready : 1              0x80018A5C
>
```

- 4.2. 当 loop 线程在控制台 1 中执行，并且在控制台 2 中执行 suspend 命令时，为什么控制台 1 中的 loop 线程处于就绪状态而不是运行状态？

答：

当在控制台 2 中执行 suspend 命令时，优先级为 24 的控制台 2 线程抢占了处理器，控制台 2 线程处于运行状态，所以此时 loop 线程处于就绪状态。

- 4.3. 实验 3.2 节中只调试了图 5-3 中显示的最主要的四种转换过程，对于线程由新建进入就绪状态，或者由任意状态进入结束状态的转换过程还没有调试，请读者找到这两个转换过程执行的源代码，自己练习调试。

答：

- 4.4. 总结一下在图 5-3 中显示的转换过程，哪些需要使用线程控制块中的上下文（将线程控制块中的文恢复到处理器中，或者将处理器的状态复制到线程控制块的上下文中），哪些不需要使用，并说明原因。

答：

就绪到运行，运行到就绪，以及运行到阻塞需要使用 TCB，因为这些过程有线程调进或者调出处理机；新建到就绪，阻塞到就绪不需要 TCB 上下文，因为没有占用处理机资源。进程在运行过程中或执行系统调用，或产生了一个中断事件，处理器都进行一次模式切换。

- 4.5. 在本实验 3.2 节中总结的所有转换过程都是分步骤进行的，为了确保完整



性，显然这些转换过程是不应该被打断的，也就是说这些转换过程都是原语操作（参见本书第 2.6 节）。请读者找出这些转换过程的原语操作（关中断和开中断）是在哪些代码中完成的。

答：

```
IntState=KeEnableInterrupts (FALSE); //关中断
KeEnableInterrupts(IntState); //开中断
```

- 4.6. 修改 EOS 源代码，对已经实现的线程的挂起状态进行改进。首先，不再使用 Zero 状态表示静止就绪状态，在枚举类型 THREAD\_STATE 中定义一个新的项用来表示静止就绪状态，并对 PsSuspendThread 函数进行适当修改。其次，处于阻塞状态和运行状态的线程也应该可以被挂起并被恢复，读者可以参考第 5.2.4 节中的内容以及图 5-5 来完成此项改进。注意，处于运行状态的线程可以将自己挂起。要设计一些方案对所修改的代码进行全面的测试，保证所做的改进是正确的。如果完成了以上改进，请思考一下控制台命令 pt 需要进行哪些相应的修改？

STATUS

PsResumeThread(

    IN HANDLE hThread

)

{

    STATUS Status;

    BOOL IntState;

    PTHREAD Thread;

    Status=ObRefObjectByHandle(hThread, PspThreadType, (PVOID\*)&Thread;

    if(EOS\_SUCCESS(Status)) {

        IntState=KeEnableInterrupts (FALSE); //关中断

        (Zero==Thread->State) {

            ListRemoveEntry(&Thread->StateListEntry);

            PspReadyThread(Thread);

```

        PspThreadSchedule();
        Status=STATUS_SUCCESS;

    }else{

        Status=STATUS_NOT_SUPPORTED;

    }

    KeEnableInterrupts(IntState); //开中断
    ObDerefObject(Thread);

}

return Status;

}

```

## 5. 总结和感想体会

本次实验学会了调试线程在各种状态间的转换，熟悉了线程的状态和转换。明白了 EOS 为线程添加的挂起状态，以及 Suspend 和 Resume 原语操作。通过逐步调试观察，明白了控制台派遣线程的阻塞、就绪与运行。通过执行 suspend 命令挂起 loop 线程的方法对 PsSuspendThread 函数进行调试，加深了对 Resume 原语的理解，学会了自己修改部分 Resume 原语。